Package 'validate'

August 7, 2017

Maintainer Mark van der Loo <mark.vanderloo@gmail.com>

License GPL-3

Title Data Validation Infrastructure

LazyData no

Type Package

LazyLoad yes

Description Declare data validation rules and data quality indicators; confront data with them and analyze or visualize the results. The package supports rules that are per-field, in-record, cross-record or cross-dataset. Rules can be automatically analyzed for rule type and connectivity.

Version 0.2.0

Depends R (>= 3.1.3), methods

URL https://github.com/data-cleaning/validate

BugReports https://github.com/data-cleaning/validate/issues

Date 2017-08-02

Imports graphics, settings, yaml

Suggests testthat, knitr, rmarkdown

Enhances lumberjack

VignetteBuilder knitr

Collate 'rule.R' 'sugar.R' 'validate_pkg.R' 'parse.R' 'expressionset.R' 'indicator.R' 'validator.R' 'confrontation.R' 'barplot.R' 'compare.R' 'factory.R' 'functions.R' 'lumberjack.R' 'retailers.R' 'utils.R' 'yaml.R'

RoxygenNote 6.0.1

NeedsCompilation yes

Author Mark van der Loo [cre, aut], Edwin de Jonge [aut], Paul Hsieh [ctb]

Repository CRAN

Date/Publication 2017-08-07 18:53:12 UTC

R topics documented:

+,indicator,indicator-method

Combine two indicator objects

Description

Combine two indicator objects by addition. A new indicator object is created with default (global) option values. Previously set options are ignored.

Usage

```
## S4 method for signature 'indicator,indicator'
e1 + e2
```

Arguments

e1	a validator
e2	a validator

Examples

```
indicator(mean(x)) + indicator(x/median(x))
```

+,validator,validator-method

Combine two validator objects

Description

Combine two validator objects by addition. A new validator object is created with default (global) option values. Previously set options are ignored.

Usage

```
## S4 method for signature 'validator,validator'
e1 + e2
```

Arguments

e1	a validator
e2	a validator

Note

The names of the resulting object are made unique using make.names.

Examples

```
validator(x>0) + validator(x<=1)
```

`aggregate,validation-method`

Aggregate validation results

Description

Aggregate results of a validation.

Usage

```
## S4 method for signature 'validation'
aggregate(x, by = c("rule", "record"), drop = TRUE,
  ...)
```

Arguments

`x`	An object of class `validation`
`by`	Report on violations per rule (default) or per record?
`drop`	drop list attribute if the result is list of length 1
`...`	Arguments to be passed to or from other methods.

Value

By default, a `data.frame` with the following columns.

`npass`	Number of items passed
`nfail`	Number of items failing
`nNA`	Number of items resulting in `NA`
`rel.pass`	Relative number of items passed
`rel.fail`	Relative number of items failing
`rel.NA`	Relative number of items resulting in `NA`

If `by='rule'` the relative numbers are computed with respect to the number of records for which the rule was evaluated. If `by='record'` the relative numbers are computed with respect to the number of rules the record was tested agains.

When `by='record'` and not all validation results have the same dimension structure, a list of `data.frames` is returned.

See Also

- `summary,validation-method`

- barplot,validation-method
- sort,validation-method
- validation

Examples

```
data(retailers)
retailers$id <- paste0("ret",1:nrow(retailers))
v <- validator(
    staff.costs/staff < 25
  , turnover + other.rev==total.rev)

cf <- confront(retailers,v,key="id")
a <- aggregate(cf,by='record')
head(a)

# or, get a sorted result:
s <- sort(cf, by='record')
head(s)
```

`as.data.frame` *Coerce to* `data.frame`

Description

Coerce to `data.frame`

Usage

```
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	Object to coerce
`row.names`	ignored
`optional`	ignored
`...`	arguments passed to other methods

`as.data.frame,confrontation-method`
Confrontation object to data frame

Description

Confrontation object to data frame

Usage

```
## S4 method for signature 'confrontation'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)
```

Arguments

`x`	Object to coerce
`row.names`	ignored
`optional`	ignored
`...`	arguments passed to other methods

Value

A `data.frame` with columns

- `key` Where relevant, and only if `key` was specified in the call to `confront`
- `name` Name of the rule
- `value` Value after evaluation
- `expression` evaluated expression

Examples

```
cf <- check_that(women, height > 0, sd(weight) > 0)
as.data.frame(cf)

# add id-column
women$id <- letters[1:15]
i <- indicator(mw = mean(weight), ratio = weight/height)
as.data.frame(confront(women, i, key="id"))
```

as.data.frame,expressionset-method

Translate an expressionset to data.frame

Description

Expressions are deparsed and combined in a `data.frame` with (some of) their metadata. Observe that some information may be lost (e.g. options local to the object).

Usage

```
## S4 method for signature 'expressionset'
as.data.frame(x, expand_assignments = TRUE, ...)
```

Arguments

`x`	Object to coerce
`expand_assignments`	Toggle substitution of ':=' assignments.
`...`	arguments passed to other methods

Value

A `data.frame` with elements `rule`, `name`, `label`, `origin`, `description`, and `created`.

barplot,validation-method

Plot number of violations

Description

Plot number of violations

Usage

```
## S4 method for signature 'validation'
barplot(height, ..., order_by = c("fails", "passes",
  "nNA"), stack_by = c("fails", "passes", "nNA"), topn = Inf,
  add_legend = TRUE, add_exprs = TRUE, colors = c(fails = "#FC8D59",
  passes = "#91CF60", nNA = "#FFFFBF"))
```

Arguments

height	an R object defining height of bars (here, a validation object)
...	parameters to be passed to barplot but not height, horiz, border,las, and las.
order_by	(single character) order bars decreasingly from top to bottom by the number of fails, passes or NA's.
stack_by	(3-vector of characters) Stacking order for bar chart (left to right)
topn	If specified, plot only the top n most violated calls
add_legend	Display legend?
add_exprs	Display rules?
colors	Bar colors for validations yielding NA or a violation

Value

A list, containing the bar locations as in barplot

Credits

The default colors were generated with the RColorBrewer package of Erich Neuwirth.

See Also

- summary,validation-method
- aggregate,validation-method
- sort,validation-method
- validation

Examples

```
data(retailers)
cf <- check_that(retailers
    , staff.costs < total.costs
    , turnover + other.rev == total.rev
    , other.rev > 0
    , total.rev > 0)
barplot(cf)
```

cells *Cell counts and differences for a series of datasets*

Description

Cell counts and differences for a series of datasets

Usage

```
cells(..., .list = NULL, compare = c("to_first", "sequential"))
```

Arguments

...	A (named) sequence of R objects carrying data (*e.g.* data.frames)
.list	A list of R objects carrying data; will be concatenated with objects in ...
compare	How to compare the datasets.

Value

An object of class cellComparison, which is really an array with a few attributes. It counts the total number of cells, the number of missings, the number of altered values and changes therein as compared to the reference defined in how.

Details

This function assumes that the datasets have the same dimensions and that both rows and columns are ordered similarly.

See Also

- compare
- match_cells

Examples

```
data(retailers)

# start with raw data
step0 <- retailers

# impute turnovers
step1 <- step0
step1$turnover[is.na(step1$turnover)] <- mean(step1$turnover,na.rm=TRUE)

# flip sign of negative revenues
step2 <- step1
step2$other.rev <- abs(step2$other.rev)
```

```
# create an overview of differences compared to raw data
cells(raw = step0, imputed = step1, flipped = step2)

# create an overview of differences, comparing to the previous step
cells(raw = step0, imputed = step1, flipped = step2, compare="sequential")
```

---

check_that *Simple data validation interface*

Description

Simple data validation interface

Usage

```
check_that(dat, ...)
```

Arguments

dat	an R object carrying data
...	a comma-separated set of validating expressions.

Value

An object of class validation

Details

Creates an object of class validator and confronts it with the data. This function is easy to use in combination with the **magrittr** pipe operator.

Examples

```
cf <- check_that(women, height>0, height/weight < 0.5)
cf
summary(cf)
barplot(cf)


## Not run:
# this works only after loading the 'magrittr' package
women %>%
  check_that(height>0, height/weight < 0.5) %>%
  summary()

## End(Not run)
```

compare *Compare similar data sets*

Description

Compare different versions of the same dataset with respect to predifined indicators. Results are simplified in a sensible way.

Usage

```
compare(x, ...)

## S4 method for signature 'validator'
compare(x, ..., .list = NULL, how = c("to_first",
  "sequential"))

## S4 method for signature 'indicator'
compare(x, ..., .list = NULL)
```

Arguments

x	An R object
...	(named) data sets (*e.g.* data.frames)
.list	Optional list of data sets, will be concatenated with ....
how	how to compare

Value

For validator: An array where each column represents one dataset. The rows count the following attributes:

- Number of validations performed
- Number of validations that evaluate to NA (unverifiable)
- Number of validations that evaluate to a logical (verifiable)
- Number of validations that evaluate to TRUE
- Number of validations that evaluate to FALSE
- Number of extra validations that evaluate to NA (new unverifiable)
- Number of validations that still evaluate to NA (still unverifialble)
- Number of validations that still evaluate to TRUE
- Number of extra validations that evaluate to TRUE
- Number of validations that still evaluate to FALSE
- Number of extra validations that evaluate to FALSE

For indicator: A list with the following components:

- `numeric`: An array collecting results of scalar indicator (e.g. `mean(x)`).
- `nonnumeric`: An array collecting results of nonnumeric scalar indicators (e.g. names(which.max(table(x))))
- `array`: A list of arrays, collecting results of vector-indicators (e.g. x/mean(x))

See Also

- `cells`
- `validator`, `validator-class`
- `indicator`, `indicator-class`

`confront` *Confront data with a (set of) expressionset(s)*

Description

Confront data with a (set of) expressionset(s)

Usage

```
confront(dat, x, ref, ...)

## S4 method for signature 'data.frame,indicator,ANY'
confront(dat, x, key = NA_character_,
  ...)

## S4 method for signature 'data.frame,indicator,environment'
confront(dat, x, ref,
  key = NA_character_, ...)

## S4 method for signature 'data.frame,indicator,data.frame'
confront(dat, x, ref,
  key = NA_character_, ...)

## S4 method for signature 'data.frame,indicator,list'
confront(dat, x, ref,
  key = NA_character_, ...)

## S4 method for signature 'data.frame,validator,ANY'
confront(dat, x, key = NA_character_,
  ...)

## S4 method for signature 'data.frame,validator,environment'
confront(dat, x, ref,
  key = NA_character_, ...)

## S4 method for signature 'data.frame,validator,data.frame'
```

```
confront(dat, x, ref,
  key = NA_character_, ...)

## S4 method for signature 'data.frame,validator,list'
confront(dat, x, ref,
  key = NA_character_, ...)
```

Arguments

dat	An R object carrying data
x	An R object carrying rules.
ref	Optionally, an R object carrying reference data. See examples for usage.
...	Options used at execution time (especially 'raise'). See voptions.
key	(optional) name of identifying variable in x.

Using reference data

When reference data sets are given, it is assumed that rows in the reference data are ordered corresponding to the rows of dat, except when a key is specified. In that case, all reference datasets are matched against the rows of dat using key Nonmatching records are removed from datasets in ref. If there are records in dat that are not in ref, then datasets in ref are extended with records containing only NA. In particular, this means that wen reference data is passed in an environment, those reference data sets may altered by the call to confront.

Technically, reference data will be stored in an environment that is the parent of a (created) environment that contains the columns of dat.

See Also

- voptions
- summary,validation-method, aggregate,validation-method, sort,validation-method
- summary,indication-method
- indicator, indicator-class
- validator, validator-class

Examples

```
# an example checking metadata
v <- validator(nrow(.) == 15, ncol(.) > 2)
summary(confront(women, v))

# An example using reference data
v <- validator(weight == ref$weight)
summary(confront(women, v, women))

# Usging custom names for reference data
v <- validator(weight == test$weight)
summary( confront(women,v, list(test=women)) )
```

```
# Reference data in an environment
e <- new.env()
e$test <- women
v <- validator(weight == test$weight)
summary( confront(women, v, e) )

# the effect of using a key
w <- women
w$id <- letters[1:nrow(w)]
v <- validator(weight == ref$weight)

# with complete data; already matching
values( confront(w, v, w, key='id'))

# with scrambled rows in reference data (reference gets sorted according to dat)
i <- sample(nrow(w))
values(confront(w, v, w[i,],key='id'))

# with incomplete reference data
values(confront(w, v, w[1:10,],key='id'))
```

`created` *Creation timestamp*

Description

Creation timestamp

Usage

```
created(x, ...)

## S4 method for signature 'rule'
created(x, ...)

## S4 method for signature 'expressionset'
created(x, ...)
```

Arguments

`x`	and R object
`...`	Arguments to be passed to other methods

Value

A `POSIXct` vector.

Methods (by class)

- expressionset: Creation time of every rule in x

See Also

- names,expressionset-method, length,expressionset-method
- description, label, origin variables

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

---

created<- *Set creation timestamp*

Description

Set creation timestamp

Usage

```
created(x) <- value
```

Arguments

x	Object
value	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

---

created<-,expressionset,POSIXct-method

Set timestamps

Description

Set timestamps

Usage

```
## S4 replacement method for signature 'expressionset,POSIXct'
created(x) <- value
```

Arguments

x	Object
value	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

---

description *description description*

Description

description description

Usage

```
description(x, ...)

## S4 method for signature 'rule'
description(x, ...)
```

```
## S4 method for signature 'expressionset'
description(x, ...)
```

Arguments

x	and R object
...	Arguments to be passed to other methods

Value

A character vector.

Methods (by class)

- expressionset: description description of every rule in x

See Also

- names,expressionset-method, length,expressionset-method
- label, created, origin, variables

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

`description<-` *Set description*

Description

Set description

Usage

```
description(x) <- value
```

Arguments

x	Object
value	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

`description<-,expressionset,character-method` *Set descriptions*

Description

Set descriptions

Usage

```
## S4 replacement method for signature 'expressionset,character'
description(x) <- value
```

Arguments

x	Object
value	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

errors *Get messages from a confrontation object*

Description

Get messages from a confrontation object

Usage

```
errors(x, ...)

## S4 method for signature 'confrontation'
errors(x, ...)

## S4 method for signature 'confrontation'
warnings(x, ...)
```

Arguments

x	An object of class confrontation
...	Arguments to be passed to other methods.

See Also

- confront

Examples

```
# create an error, by using a non-existent variable name
cf <- check_that(women, hite > 0, weight > 0)
# retrieve error messages
errors(cf)
```

export_yaml *Export to yaml file*

Description

Translate an object to yaml format and write to file.

Usage

```
export_yaml(x, file, ...)

as_yaml(x, ...)

## S4 method for signature 'expressionset'
export_yaml(x, file, ...)

## S4 method for signature 'expressionset'
as_yaml(x, ...)
```

Arguments

x	An R object
file	A file location or connection (passed to base::write).
...	Options passed to yaml::as.yaml

Details

Both validator and indicator objects can be exported.

Examples

```
v <- validator(x > 0, y > 0, x + y == z)
txt <- as_yaml(v)
cat(txt)


# NOTE: you can safely run the code below. It is enclosed in 'not run'
# statements to prevent the code from being run at test-time on CRAN
## Not run:
export_yaml(v, file="my_rules.txt")

## End(Not run)
```

---

label *label description of rules*

Description

label description of rules

Usage

```
label(x, ...)

## S4 method for signature 'rule'
label(x, ...)

## S4 method for signature 'expressionset'
label(x, ...)
```

Arguments

x	and R object
...	Arguments to be passed to other methods

Value

A character vector.

Methods (by class)

- expressionset: label description of every rule in x

See Also

- names,expressionset-method, length,expressionset-method
- description, created, origin, variables

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"
```

```
# short description is also printed:
v

# print all info for first rule
v[[1]]
```

`label<-` *Set label*

Description

Set label

Usage

```
label(x) <- value
```

Arguments

`x`	Object
`value`	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
```

```
v

# print all info for first rule
v[[1]]
```

`label<-,expressionset,character-method`
Set labels

Description

Set labels

Usage

```
## S4 replacement method for signature 'expressionset,character'
label(x) <- value
```

Arguments

x	Object
value	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"
```

```
# short description is also printed:
v

# print all info for first rule
v[[1]]
```

`lbj_cells-class` *Logging object to use with the lumberjack package*

Description

Logging object to use with the lumberjack package

Format

A reference class object

Methods

`add(meta, input, output)` Add logging info based on in- and output

`dump(file = "cells.csv", ...)` Dump logging info to csv file. All arguments in '...' except row.names are passed to 'write.csv'

`initialize(..., verbose = TRUE)` Create object. Optionally toggle verbosity.

`log_data()` Return logged data as a data.frame

Details

This obeject can used with the function composition ('pipe') operator of the `lumberjack` package. The logging is based on validate's `cells` function. The output is written to a `csv` file wich contains the following columns.

`step`	`integer`	Step number
`time`	`POSIXct`	Timestamp
`expr`	`character`	Expression used on data
`cells`	`integer`	Total nr of cells in dataset
`available`	`integer`	Nr of non-NA cells
`missing`	`integer`	Nr of empty (NA) cells
`still_available`	`integer`	Nr of cells still available after expr
`unadapted`	`integer`	Nr of cells still available and unaltered
`unadapted`	`integer`	Nr of cells still available and altered
`imputed`	`integer`	Nr of cells not missing anymore

Note

This logger is suited only for operations that do not change the dimensions of the dataset.

See Also

Other loggers: lbj_rules-class

---

lbj_rules-class *Logging object to use with the lumberjack package*

Description

Logging object to use with the lumberjack package

Methods

`dump(file = "lbj_rules.csv", ...)` Dump logging info to csv file. All arguments in '...' except row.names are passed to 'write.csv'

`initialize(rules, verbose = TRUE)` Create object. Optionally toggle verbosity.

`log_data()` Return logged data as a data.frame

See Also

Other loggers: lbj_cells-class

---

length,expressionset-method *Determine the number of elements in an object.*

Description

Determine the number of elements in an object.

Usage

```
## S4 method for signature 'expressionset'
length(x)

## S4 method for signature 'confrontation'
length(x)
```

Arguments

x	An R object

match_cells *Create matching subsets of a sequence of data*

Description

Create matching subsets of a sequence of data

Usage

```
match_cells(..., .list = NULL, id = NULL)
```

Arguments

...	A sequence of data.frames, possibly in the form of <name>=<value> pairs.
.list	A list of data.frames; will be concatenated with ....
id	Names or indices of columns to use as index.

Value

A list of data.frames, subsetted and sorted so that all cells correspond.

See Also

- cells

names,expressionset-method *Extract names*

Description

Extract names

Usage

```
## S4 method for signature 'expressionset'
names(x)
```

Arguments

x	An R object

Value

A character with names of rules occurring in x

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

`names<-,expressionset,character-method`
Set names

Description

Names are recycled and made unique with `make.names`

Usage

```
## S4 replacement method for signature 'expressionset,character'
names(x) <- value
```

Arguments

`x`	Object
`value`	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

`origin` *Origin of rules*

Description

Origin of rules

Usage

```
origin(x, ...)

## S4 method for signature 'rule'
origin(x, ...)

## S4 method for signature 'expressionset'
origin(x, ...)
```

Arguments

x	and R object
...	Arguments to be passed to other methods

Value

A character vector.

Methods (by class)

- expressionset: Origin of every rule in x

See Also

- names,expressionset-method, length,expressionset-method
- description, label, created, variables

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

origin<- *Set origin*

Description

Set origin

Usage

```
origin(x) <- value
```

Arguments

x	Object
value	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

origin<-,expressionset,character-method
Set origins

Description

Set origins

Usage

```
## S4 replacement method for signature 'expressionset,character'
origin(x) <- value
```

Arguments

x	Object
value	Value to set

Examples

```
# retrieve properties
v <- validator(turnover > 0, staff.costs>0)

# number of rules in v:
length(v)

# per-rule
created(v)
origin(v)
names(v)

# set properties
names(v)[1] <- "p1"

label(v)[1] <- "turnover positive"
description(v)[1] <- "
According to the official definition,
only positive values can be considered
valid turnovers.
"

# short description is also printed:
v

# print all info for first rule
v[[1]]
```

retailers *data on retailers*

Description

Anonymized and distorted data on revenue and cost structure for 60 retailers. Currency is in thousands (of dollars or euros or guilders, ...)

- size: Size class
- incl.prob: Probability of inclusion in the sample
- staff: Number of staff
- turnover: Amount of turnover
- other.rev: Amount of other revenue
- total.rev: Total revenue
- staff.costs: Number of staff employed
- total.costs: Total costs made
- profit: Amount of profit
- vat: Turnover reported for Value Added Tax

Format

A csv file, one retailer per row.

sort,validation-method *Aggregate and sort the results of a validation.*

Description

Aggregate and sort the results of a validation.

Usage

```
## S4 method for signature 'validation'
sort(x, decreasing = FALSE, by = c("rule", "record"),
  drop = TRUE, ...)
```

Arguments

x	An object of class validation
decreasing	Sort by decreasing number of passes?
by	Report on violations per rule (default) or per record?
drop	drop list attribute if the result has a single argument.
...	Arguments to be passed to or from other methods.

Value

A `data.frame` with the following columns.

`npass`	Number of items passed
`nfail`	Number of items failing
`nNA`	Number of items resulting in `NA`
`rel.pass`	Relative number of items passed
`rel.fail`	Relative number of items failing
`rel.NA`	Relative number of items resulting in `NA`

If `by='rule'` the relative numbers are computed with respect to the number of records for which the rule was evaluated. If `by='record'` the relative numbers are computed with respect to the number of rules the record was tested agains. By default the most failed validations and records with the most fails are on the top.

When `by='record'` and not all validation results have the same dimension structure, a list of `data.frames` is returned.

See Also

- summary,validation-method
- aggregate,validation-method
- barplot,validation-method
- validation

Examples

```
data(retailers)
retailers$id <- paste0("ret",1:nrow(retailers))
v <- validator(
    staff.costs/staff < 25
  , turnover + other.rev==total.rev)

cf <- confront(retailers,v,key="id")
a <- aggregate(cf,by='record')
head(a)

# or, get a sorted result:
s <- sort(cf, by='record')
head(s)
```

summary *Create a summary*

Description

Create a summary

Usage

```
summary(object, ...)

## S4 method for signature 'expressionset'
summary(object, ...)

## S4 method for signature 'indication'
summary(object, ...)

## S4 method for signature 'validation'
summary(object, ...)
```

Arguments

object	An R object
...	Currently unused

Value

A data.frame with the information mentioned below is returned.

Validator and indicator objects

For these objects, the ruleset is split into subsets (blocks) that are disjunct in the sense that they do not share any variables. For each bloch the number of variables, the number of rules and the number of rules that are linear are reported.

Indication

Some basic information per evaluated indicator is reported: the number of items to which the indicator was applied, the output class, some statistics (min, max, mean , number of NA) and wether an exception occurred (warnings or errors). The evaluated expression is reported as well.

Validation

Some basic information per evaluated validation rule is reported: the number of items to which the rule was applied, the output class, some statistics (passes, fails, number of NA) and wether an exception occurred (warnings or errors). The evaluated expression is reported as well.

Examples

```
data(retailers)
v <- validator(staff > 0, staff.costs/staff < 20, turnover+other.revenue == total.revenue)
summary(v)

cf <- confront(retailers,v)
summary(cf)
```

`syntax` *Syntax to define validation or indicator rules*

Description

A concise overview of the `validate` syntax.

Note

This document only provides a short reference. Please refer to the vignette for worked examples: Rules in text files

Refernce the dataset as a whole

Metadata such as numer of rows, columns, column names and so on can be tested by referencing the whole data set with the '`.`'. For example, the rule `nrow(.) == 15` checks whether there are 15 rows in the dataset at hand.

Local, transient assignment

The operator '`:=`' can be used to set up local variables (during, for example, validation) to save time (the rhs of an assignment is computed only once) or to make your validation code more maintainable. Assignments work more or less like common R assignments: they are only valid for statements coming after the assignment and they may be overwritten. The result of computing the rhs is not part of a `confront`ation with data.

Groups

Often the same constraints/rules are valid for groups of variables. `validate` allows for compact notation. Variable groups can be used in-statement or by defining them with the `:=` operator.

`validator( var_group(a,b) > 0 )`

is equivalent to

`validator(G := var_group(a,b), G > 0)`

is equivalent to

`validator(a>0,b>0)`.

Using two groups results in the cartesian product of checks. So the statement

```
validator( f=var_group(c,d), g=var_group(a,b), g > f)
```

is equivalent to

```
validator(a > c, b > c, a > d, b > d)
```

File parsing

Please see the vignette on how to read rules from and write rules to file:

```
vignette("rule-files",package="validate")
```

`validate` *Data Validation Infrastructure*

Description

Data Validation Infrastructure

Introduction

Data often contain errors and missing data. A necessary step before data analysis is verifying and validating your data. Package `validate` is a toolbox for creating validation rules and checking data against these rules.

Getting started

The easiest way to get started is through the examples given in `check_that`.

The general workflow in `validate` follows the following pattern.

- Define a set of rules or quality indicator using `validator` or `indicator`.
- `confront` data with the rules or indicators,
- Examine the results either graphically or by summary.

There are several convenience functions that allow one to define rules from the commandline, through a (freeform or yaml) file and to investigate and maintain the rules themselves. Please have a look at the introductory vignette for a more thorough introduction on validation rules and the indicators vignette for an introducion on quality indicators. After you're a bit aqcuinted with the package, you will probably be interested in defining your rules separately in a text file. The vignette on rule files will get you started with that.

validation-class *Store results of evaluating validating expressions*

Description

Store results of evaluating validating expressions

Details

A object of class `validation` stores a set of results generated by evaluating an `validator` in the context of data along with some metadata.

Exported S4 methods for `validation`

- Methods exported for objects of class `confrontation`
- `summary,validation-method`
- `values,validation-method`
- `barplot,validation-method`
- `aggregate,validation-method`
- `sort,validation-method`

See also

- `confront`
- `validator`

validator *Define validation rules for data*

Description

Define validation rules for data

Usage

```
validator(..., .file, .data)
```

Arguments

`...`	A comma-separated list of validating expressions
`.file`	(optional) A character vector of file locations (see also the section on file parsing in the
`.data`	(optional) A `data.frame` with columns `"rule"`, `"name"`, and `"description"` `syntax` help file).

Value

An object of class `validator` (see `validator-class`).

Validating expressions

Each validating expression should evaluate to a `logical`. Allowed syntax of the expression is described in `syntax`.

A *validating expression* is an expression whose evaluation results in `TRUE`, `FALSE` or `NA`.

See Also

- `syntax`
- `confront`, `check_that`
- `summary,expressionset-method`
- `validator-class`

Examples

```
v <- validator(
  height>0
  ,weight>0
  ,height < 1.5*mean(height)
)
cf <- confront(women, v)
summary(cf)
```

`values` *Get values from object*

Description

Get values from object

Usage

```
values(x, ...)

## S4 method for signature 'confrontation'
values(x, ...)

## S4 method for signature 'validation'
values(x, simplify = TRUE, drop = TRUE, ...)

## S4 method for signature 'indication'
values(x, simplify = TRUE, drop = TRUE, ...)
```

Arguments

x	an R object
...	Arguments to pass to or from other methods
simplify	Combine results with similar dimension structure into arrays?
drop	if a single vector or array results, drop 'list' attribute?

---

variables *Extract variable names*

Description

Extract variable names

Usage

```
variables(x, ...)

## S4 method for signature 'rule'
variables(x, ...)

## S4 method for signature 'list'
variables(x, ...)

## S4 method for signature 'data.frame'
variables(x, ...)

## S4 method for signature 'environment'
variables(x, ...)

## S4 method for signature 'expressionset'
variables(x, as = c("vector", "matrix", "list"),
  dummy = FALSE, ...)
```

Arguments

x	An R object
...	Arguments to be passed to other methods.
as	how to return variables: • 'vector' Return the uniqe vector of variables occurring in x. • 'matrix' Return a boolean matrix, each row representing a rule, each column representing a variable. • 'list' Return a named list, each entry containing a character vector with variable names.
dummy	Also retrieve transient variables set with the := operator.

Methods (by class)

- `rule`: Retrieve unique variable names
- `list`: Alias to `names.list`
- `data.frame`: Alias to `names.data.frame`
- `environment`: Alias to `ls`
- `expressionset`: Variables occuring in `x` either as a single list, or per rule.

See Also

- `names,expressionset-method`, `length,expressionset-method`
- `description`, `label`, `created`, `origin`

Examples

```
v <- validator(
  root = y := sqrt(x)
 , average = mean(x) > 3
 , sum = x + y == z
)
variables(v)
variables(v,dummy=TRUE)
variables(v,matrix=TRUE)
variables(v,matrix=TRUE,dummy=TRUE)
```

---

voptions *Set or get options globally or per object.*

Description

Set or get options globally or per object.

Usage

```
voptions(x = NULL, ...)

## S4 method for signature 'ANY'
voptions(x = NULL, ...)

validate_options(...)

reset(x = NULL)

## S4 method for signature 'ANY'
```

```
reset(x = NULL)

## S4 method for signature 'expressionset'
voptions(x = NULL, ...)

## S4 method for signature 'expressionset'
reset(x = NULL)
```

Arguments

x	(optional) an object inheriting from expressionset such as validator or indicator.
...	Name of an option (character) to retrieve options or option = value pairs to set options.

Value

When requesting option settings: a list. When setting options, the whole options list is returned silently.

Options for the validate package

Currently the following options are supported.

- na.value (NA,TRUE,FALSE; NA) Value to return when a validating statement results in NA.
- raise ("none","error","all"; "none") Control if the confront methods catch or raise exceptions. The 'all' setting is useful when debugging validation scripts.
- lin.eq.eps ('numeric'; 1e-8) The precision used when evaluating linear equalities. To be used to control for machine rounding.
- "reset" Reset to factory settings.

Details

There are three ways in which options can be specified.

- Globally. Setting voptions(option1=value1,option2=value2,...) sets global options. Per object. Setting voptions(where=<object>, option1=value1,...), causes all relevant functions that use that object (e.g. confront) to use those local settings. At execution time. Relevant functions (e.g. confront) take optional arguments allowing one to define options to be used during the current function call

To set options in a file, use voptions(option1=value1,option2=value2,...) without the where argument. This will invoke a local setting in the object created when the file is parsed.

Examples

```
# the default allowed validation symbols.
voptions('validator_symbols')

# set an option, local to a validator object:
v <- validator(x + y > z)
```

```
voptions(v,raise='all')
# check that local option was set:
voptions(v,'raise')
# check that global options have not changed:
voptions('raise')
```

`[,expressionset-method`
Select a subset

Description

Select a subset

Usage

```
## S4 method for signature 'expressionset'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'expressionset'
x[[i, j, ..., exact = TRUE]]

## S4 method for signature 'confrontation'
x[i, j, ..., drop = TRUE]
```

Arguments

`x`	An R object
`i`	an index (numeric, boolean, character)
`j`	not implemented
`...`	Arguments to be passed to other methods
`drop`	not implemented
`exact`	Not implemented

Value

An new object, of the same class as `x` subsetted according to `i`.

Details

The `options` attribute will be cloned

Index